Software Engineering SS18

Course Exercises and Material

April 30, 2018

1 Intoduction

In industry, the devoloper who implements a module is not always the same as the one that tests it. This allows for more unbiased test of the specification. When a single developer plays both of these roles, the developer may write a test suite that unwittingly takes advantages of knowledge of the implementation, and thus fails to catch errors. Separating these roles solves this problem.

In this problem set you play the role of the test designer. Your task is to design a test suite for a module based on the specification of the module alone. More precisely, you will have to test our implementation of a Graph, whose specification is a simplified version of JGraphT (https://github.com/lingeringsocket/jgrapht).

2 Extras

Before you begin, please think about this small related problems:

1. Look at the specification for Math.max(int a, int b). Identify the input subdomains and list the combinations of a and b that you would test if you were to write a test method for this method. Keep boundary cases in mind.

2. Suppose a method `d(Object obj)` specifies that it throws an `IllegalArgumentException` whenever obj is `null`. Someone writes the following JUnit test:

```
try {
  d(null)
} catch (Exception e) {
  assertTrue("Exceptions was thrown.", true);
}
```

   Unfortunately, this test does not verify that `d()` meets its specification. Why is this the case?

3 Problem

Using solely the specification of for the Graph class, design a JUnit suite that tests an implementation of Graph. Use it to test the specific implementation of the Graph class provided.

The Graph class should act as specified; however the implementation provided contains errors. You should identify and document as many bugs as possible.

Note, your test suite should not be tailored to a specific kind of implementation. Ie. it should not produce false errors when applied to a correct implementation, and it should be capable of revealing different bugs in other implementations.

When designing your test suite, you should keep the following points in mind:

- The suite should be compact, meaning you should not include gratuitous tests. You should make it as small as you can, while still achieving good code coverage.
- The suite should be informative, meaning that once your test suite finds a bug in the module, it should give enough information about the bug so that someone else could identify and fix the bug in the module quickly.
- The suite should be thorough, meaning that if an implementation of the specification passes your test suite, then you can be reasonably sure that it will always adhere to its specification in practice.

4 Hints

- You'll save a lot of time, and produce a much smaller and more effective test suite, by selecting your input subdomains first, and only then choosing test cases.
- Use appropriate names for your methods and descriptive text in your failure messages. Note that every assert method in junit.framework.TestCase has two sets of parameters: one that takes a String message as its first parameter and one that does not. Use the String message to describe the nature of the error that the test finds.
- Make sure that your test validates a working implementation of `Graph`. For example, you may accidentally write the following:

```
public void testFoo() {
 methodThatRevealsProblemWithFoo();
 assertEquals(true, false); // this will always fail!
}
```

When you run this test on an implementation where foo() is broken, then it will fail as expected. However, if you run it on a correct implementation of foo(), then it will also fail because true is not equal to false.